VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

STOLNÍ HRA DÁMA

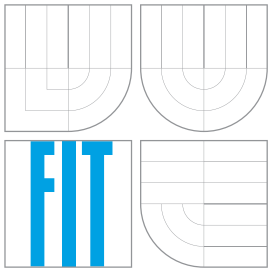
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ STANĚK

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

STOLNÍ HRA DÁMA

DESKTOP GAME “DÁMA”

BAKALÁŘSKÁ PRÁCE
BACHELOR’S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ STANĚK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. DUŠAN KOLÁŘ

BRNO 2013

Abstrakt

V této práci je implementována stolní hra dáma. Aplikace umožňuje hrát podle různých pravidel dámy. Podporuje různé způsoby vyhledávání možných tahů hráče. Jedním ze způsobů je paralelní výpočet.

Abstract

This bachelor's thesis implements the draughts desktop game. Application allows to play according to different rules. It supports different ways of possible turns search of the player. One of them is the parallel computation.

Klíčová slova

dáma, hra, alfabeta, minimax, paralelizace, java

Keywords

checkers, draughts, game, alphabeta, minimax, parallelism, java

Citace

Ondřej Staněk: Stolní hra Dáma, bakalářská práce, Brno, FIT VUT v Brně, 2013

Stolní hra Dáma

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Dr. Ing. Dušana Koláře

.....
Ondřej Staněk
14. května 2013

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Doc. Dr. Ing. Dušanovi Koláři za cenné rady a připomínky při řešení této práce. Dále bych chtěl poděkovat své rodině za podporu.

Obsah

Kapitola 1

Úvod

V této práci je řešena implementace stolní hry dáma. Cílem práce je vytvořit implementaci, která bude funkční a jednoduchá na pochopení. Aplikace je opensource, proto je důležité, aby kód aplikace byl přehledný, dobře objektově navržený a snadno rozšiřitelný. Program umožňuje hrát hru podle řady známých pravidel, umožňuje ale také uživateli vytvořit si vlastní pravidla podle parametrů daných nastavením hry. Je možné zvolit si typ hry, kdy bude uživatel hrát proti počítači nebo proti člověku. Pro demonstraci algoritmů hry hra obsahuje i možnost hry dvou počítačů proti sobě. Hra dává možnost nastavit si, které algoritmy bude počítač používat pro výpočet. Přičemž v aplikaci existuje i varianta, která používá při výpočtu paralelizaci.

Kapitola 2

Pravidla hry dáma

Hra dáma se hraje na čtvercové desce zpravidla o velikosti 8x8 nebo 10x10 polí, přičemž se pravidelně střídají světlá a tmavá pole. Hraje se pouze na aktivních tmavých polích. Každý hráč má určitý počet kamenů buď bílé, nebo černé barvy, které jsou rozestavěny na konci hracího pole proti sobě. Figury se rozlišují na kameny a dámy, které mají různý způsob pohybu a braní. Kámen se pohybuje dopředu po diagonále o jedno pole. Braní soupeřovy figury probíhá tak, že kámen přeskočí danou figuru, která je před ním, umístí se za figuru a soupeřova figura je odstraněna z hrací desky. Braní kamenem je možné pouze ve směru po diagonále, a pokud je za soupeřovou figurou volné pole. Braní soupeřovy figury je povinné. Vícenásobné braní se považuje za jeden tah.

Kámen je vyměněn za dámu v okamžiku, kdy se dostane na protilehlou základní řadu hrací desky. Pohyb dámy je různý podle typu pravidel. Dáma se může pohybovat po diagonále všemi směry buď o jedno pole, nebo o libovolný počet polí v závislosti na pravidlech hry. Braní dámou může být také různé. Dáma podle většiny pravidel nemusí při braní skákat za soupeřovu figuru, existují ale i pravidla, kdy dáma za figuru skákat musí. [1]

2.1 Česká dáma

Podle pravidel české dámy se hraje na poli o velikosti 8x8 polí. Na hracím poli jsou umístěny tři řady kamenů pro každého hráče, celkem 12 kamenů pro každého hráče. Dáma se může na rozdíl od kamene pohybovat všemi směry a o libovolný počet polí. Pokud hráč může skákat kamenem a zároveň dámou, tak musí skákat dámou. Pokud dáma skáče, tak nemusí skákat za přeskakovanou figuru, ale může dokončit skok na kterémkoliv poli za přeskočeným kamenem.

2.2 Mezinárodní dáma

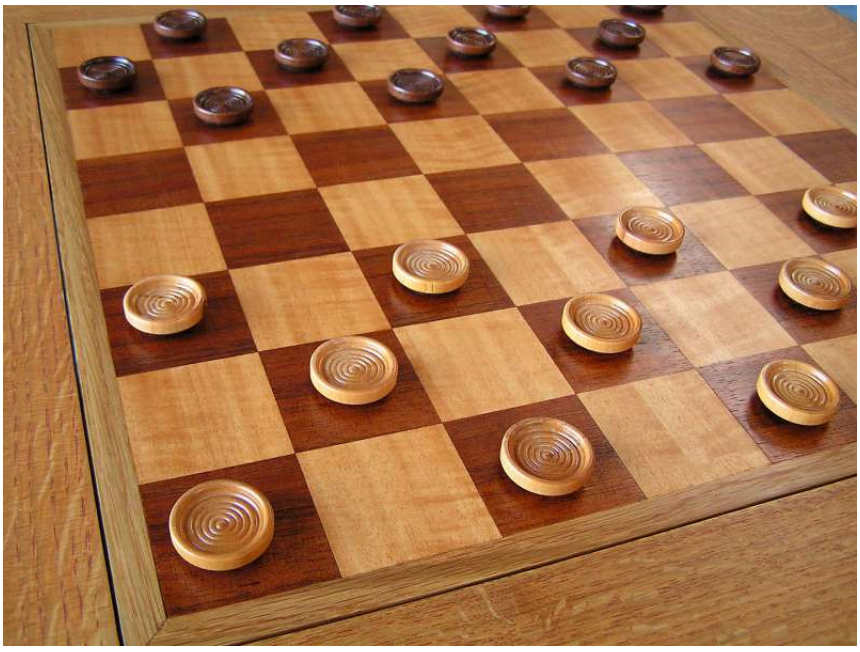
Mezinárodní dáma je nejvýznamnější varianta dámy. Hraje podle ní Světová federace dámy, jejíž oficiální francouzský název je Federation Mondiale du Jeu de Dames – FMJD. Byla založena roku 1947 a má v současné době 49 členských zemí z Evropy, Asie, obou Amerik a Afriky. Nejvíce hráčů je podobně jako v šachu organizováno v zemích bývalého SSSR, zejména v Rusku, Bělorusku a pobaltských státech.

Hraje se na desce 10x10 polí, každý hráč má 4 řady kamenů, celkem 20 kamenů pro každého hráče. Rozdíl oproti pravidlům české dámy je v tom, že kameny se skáče dopředu i dozadu. Přednost má skok, který přeskakuje větší počet figur. Dáma nemá při skákání přednost před kamenem. Dáma opět nemusí skákat za přeskakovanou figuru protihráče.

2.3 Anglická dáma

Jedná se o druhou nejvýznamnější variantu dámy, hned po mezinárodní dámě. Evropě ji nazývají draughts, v Americe jí říkají checkers. Varianta nemá celosvětovou organizaci, ale její funkci zastupují národní federace, jimiž jsou ACF (American Checkers Federation) a BDF (British Draughts Federation).

Dáma se na rozdíl od české dámy nemůže pohybovat o libovolný počet polí, ale pouze o jedno pole. Dáma se v anglické dámě nazývá král. Pokud hráč může skákat kamenem a zároveň králem, tak si může vybrat, kterou figurou chce skákat. Král nemá přednost při skákání a neplatí žádná pravidla přednosti při skákání. [5]



Obrázek 2.1: Ukázka české dámy
[1]

[1] http://img.phoneky.com/symbian-games/preview/s5/Games/3/s/1307800710-2.jpg

Kapitola 3

Teorie her pro dva hráče

Pro řešení úloh teorie her se používá rozklad obtížnějších úloh na jednodušší podproblémy. Existují dva typy podproblémů. Jedná se o problém OR, který je řešitelný, pokud je alespoň jeden z jeho podproblémů řešitelný a problém AND, který je řešitelný, pokud jsou všechny jeho podproblémy řešitelné. S těmito dvěma typy podproblémů pracují AND/OR grafy.

Uvedeného způsobu se využívá při hře dvou hráčů, protihráčů, kteří se po odehrání svých tahů pravidelně střídají. Oba hráči mají všechny informace o stavu hry, neporušují pravidla a oba se snaží zvítězit.

Problém hledání nejlepšího tahu hráče A (hráč, který je na tahu) je řešitelný, pokud alespoň jeden z jeho tahů, povede k jeho výhře (problém OR). Při hledání následují tahy hráče B, které musí být pro hráče A všechny řešitelné (problém AND). Jakmile hráč A vybere svůj nejlepší tah, tak jej odehraje a celé dosavadní prohledávání zapomene. Poté hraje hráč B, který opět hledá svůj nejlepší tah.

Hry pro dva hráče můžeme rozdělit na jednoduché hry, složité hry a hry s neurčitostí. Z těchto her se podrobně popíšu metody pro složité hry, mezi které patří i hra dáma.

3.1 Jednoduché hry

Jednoduché hry jsou hry, u kterých můžeme prohledat celý AND/OR graf v reálném čase. Nemají tolik možností tahů, proto je výpočet dostatečně rychlý. Výpočet hledání tahů u jednoduché hry se používá např. u hry zápalky, kdy hráči střídavě odebírají z hromádky zápalek. Mohou odebrat jednu, dvě nebo tři zápalky. Prohrává hráč, který nemá, co odebrat.

3.2 Hry s neurčitostí

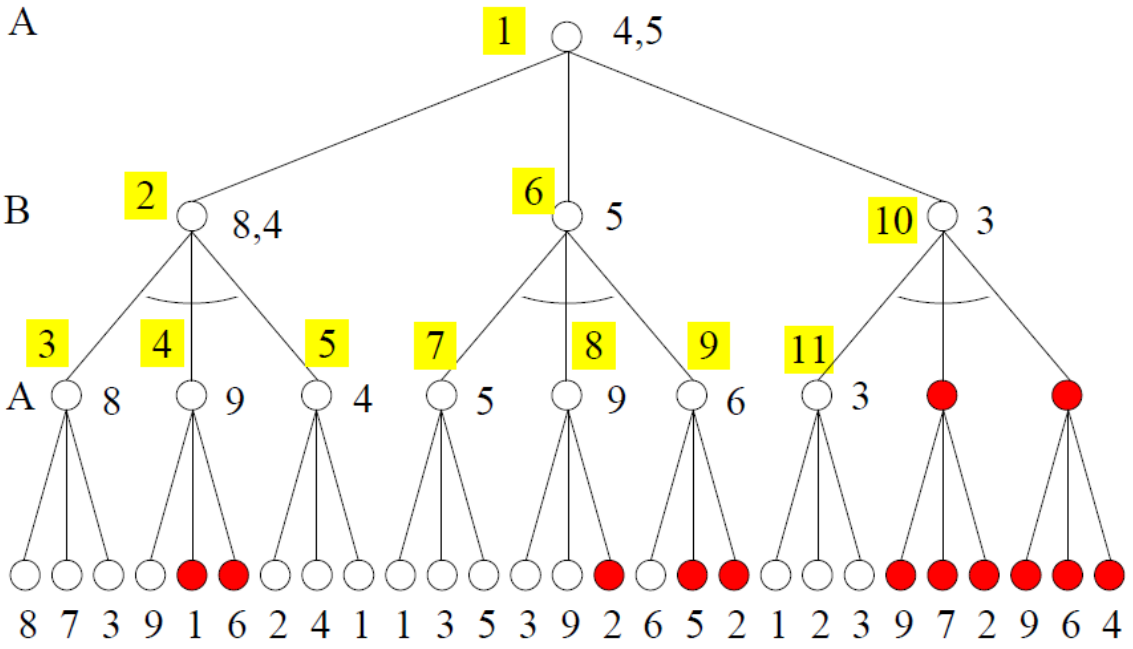
Hrám s neurčitostí se nebudu příliš věnovat, protože s problematikou dámy příliš nesouvisí. Jde o hry, ve kterých se hraje s hrací kostkou, tzn., je u nich uplatněna náhodnost. Jedná se o rozšíření metody pro složité hry, které nyní popíšu.

3.3 Složité hry

Složité hry jsou hry, mezi které patří např. dáma nebo šachy. U těchto her nemohu prohledávat celý AND/OR graf, protože obsahuje příliš velké množství tahů. Používá se algoritmu prohledávání do předem dané hloubky. Uzly v této hloubce označíme jako koncové a vrátíme jejich ohodnocení. K tomu se používá ohodnocovací funkce, která v závislosti na různých kritériích ohodnotí koncový uzel/stav hry a vrátí odpovídající skóre. Hráč A si vybírá maximální hodnoty ohodnocení, hráč B si vybírá minimální hodnoty ohodnocení. Podle toho vznikl název základního algoritmu Minimax.

3.4 Algoritmus Minimax

Základem algoritmu je rekurzivní funkce, která se volá pro aktuální stav hry (kořenový uzel AND/OR grafu). Funkce vrací ohodnocení uzlu a tah s doposud nalezeným maximálním ohodnocením. Algoritmus probíhá následovně. Pokud je prohledávaný uzel listem (bylo dosáhnuto maximální hloubky nebo je konečným stavem hry), tak se volá ohodnocovací funkce a je vráceno ohodnocení uzlu. Jinak: Je-li na tahu hráč A, tak se volá funkce Minimax pro každý tah, který je možné vykonat z aktuálního stavu hry a je vrácena maximální z vrácených hodnot. Je-li na tahu hráč B, tak je volána funkce Minimax pro každý jeho tah a je vrácena minimální z vrácených hodnot.



Obrázek 3.1: Příklad na zbytečná vyšetřování některých uzlů AND/OR grafu metodou MiniMax

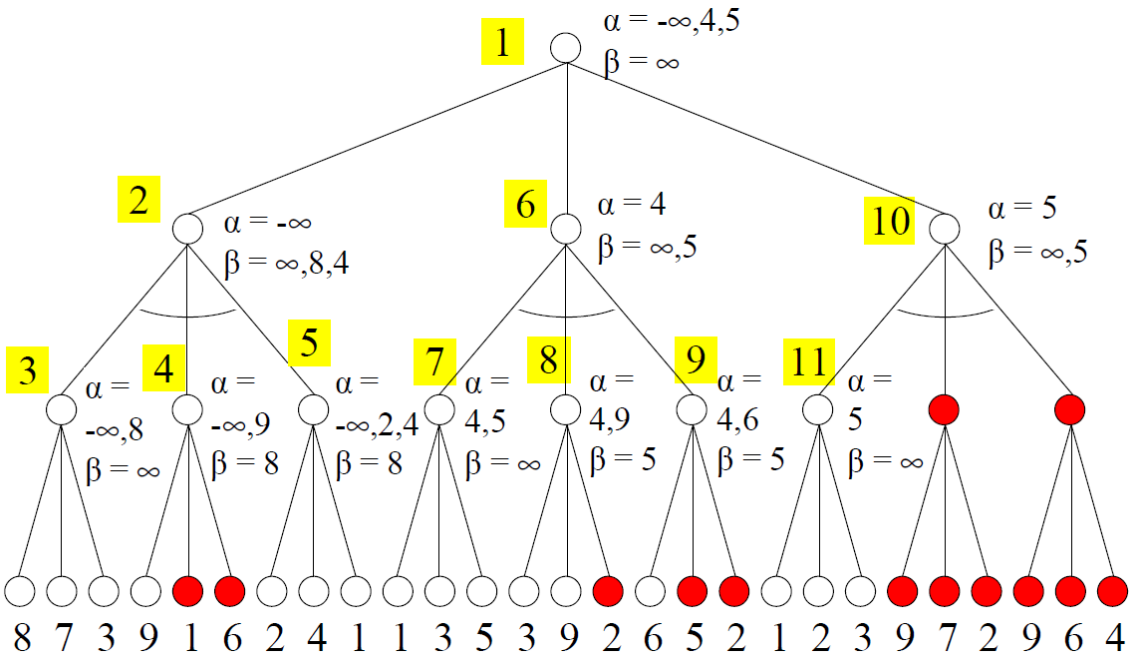
[8]

3.5 Algoritmus Alfa a Beta řezy

Optimalizuje algoritmus Minimax. Nevyšetřuje tahy, které jsou pro hráče nevýhodné a které si hráč v žádném případě nevybere. Tím ušetří čas zbytečným procházením uzlů. Alfa řezy zabrání zbytečnému vyšetřování uzlů hráče A, Beta řezy zbytečnému vyšetřování uzlů hráče B.

Metoda AlfaBeta

Pro kořenový uzel nastavím hodnotu α na minimální možnou hodnotu a hodnotu β na maximální možnou hodnotu. Pokud je prohledávaný uzel listem (bylo dosáhnuto maximální hloubky nebo je konečným stavem hry), tak se volá ohodnocovací funkce a je vráceno ohodnocení uzlu. Je-li na tahu hráč A, volám metodu AlfaBeta, dokud je $\alpha < \beta$. Po každém vyšetřeném tahu nastavím hodnotu α na maximum z aktuální a navrácené hodnoty. Pokud je $\alpha > \beta$, tak ukončím metodu a vrátím aktuální hodnotu proměnné α a nejlépe ohodnocený tah. Je-li na tahu hráč B, volám metodu AlfaBeta, dokud je $\alpha < \beta$. Po každém vyšetřeném tahu nastavím hodnotu β na minimum z aktuální a navrácené hodnoty. Pokud je $\alpha > \beta$, tak ukončím funkci a vrátím aktuální hodnotu proměnné α a nejlépe ohodnocený tah. [8]



Obrázek 3.2: Příklad práce metody AlfaBeta (alfa a beta řezů) [8]

Kapitola 4

Dostupné open-source implementace hry na počítači

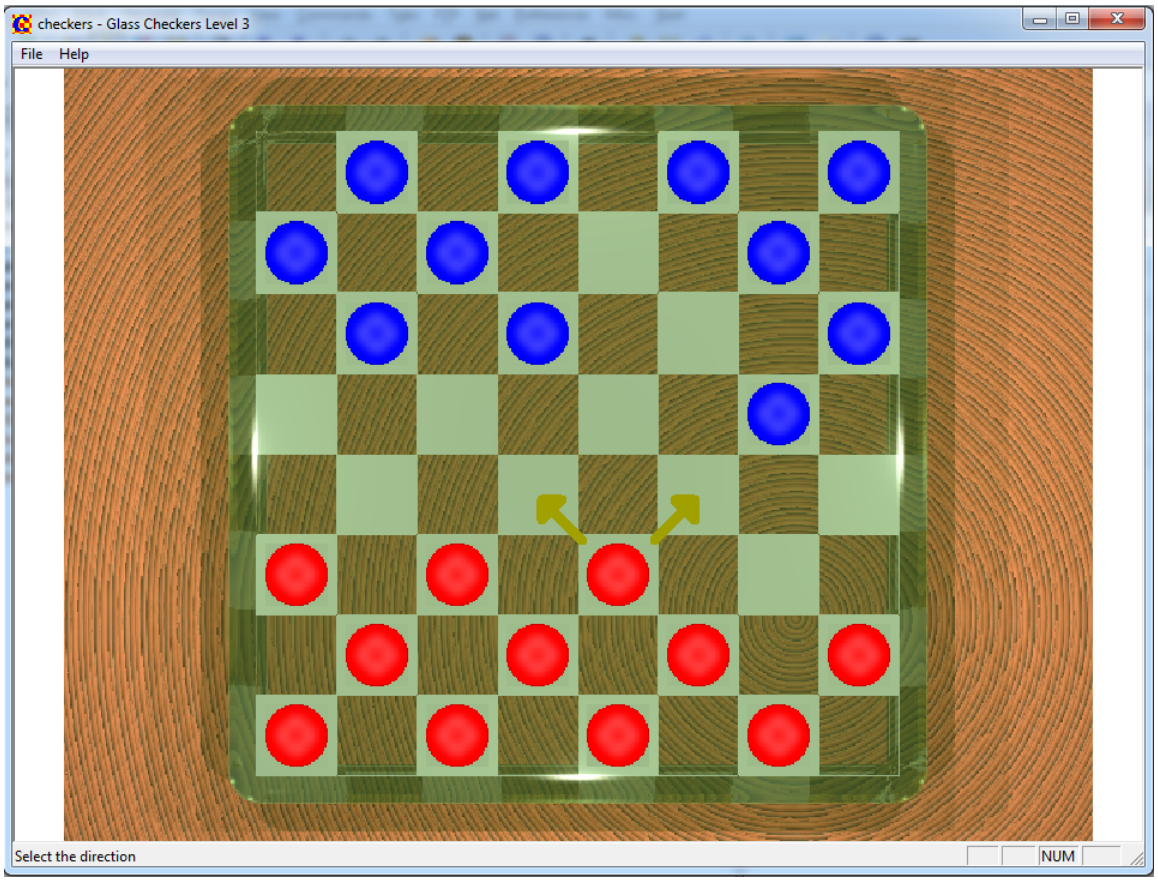
Studoval jsem následující open-source implementace dámy dostupné na internetu:

- [1]Implementace dámy, kde hrají dva lidé proti sobě. Umí hlídat dodržování pravidel. Hráč typu počítač není implementován. Hra je dobře a přehledně popsaná. Je možné ji hrát online. Zajímavé je zobrazení možných tahů hrajícího hráče.
- [2]Jednoduchá funkční implementace v Javě. Používá ohodnocovací funkci, která počítá počet figur hráče a protihráče. Jedná se o nejzákladnější typ heuristiky, který je nicméně funkční. Nelíbilo se mi, že implementace obsahuje velmi málo komentářů a kód se tak stává trochu nepřehledným a špatně pochopitelným.
- [3]Pokročilejší implementace v jazyku C++. Oproti materiálnímu ohodnocení navíc ohodnocuje pozice figur. Je možná pouze hra člověka proti počítači.
- Glass Checkers[4] – Implementace dámy v jazyku C++. Bohužel pouze pro operační systém Windows. Ohodnocovací funkce počítá s počtem figur hráčů i s pozicemi. Je možné hrát hru po síti. Program má některé části okomentované. Důležité části aplikaci by však mohly být popsány detailněji. Grafika této aplikace se podle mého názoru docela povedla. Její ukázku můžete vidět na obrázku 4.1.

Bohužel většina kvalitních implementací hry dáma není open-source. Často je možné je hrát online zdarma. Nicméně zdrojové kódy k nim dodávány nejsou.

Často jsem také narazil na implementace dámy, které byly open-source, ale jejich kód byl tak nepřehledný a často úplně bez komentářů, že jsem studium těchto implementací opustil.

[1] http://math.hws.edu/eck/cs124/javanotes3/c8/s5.html
[2] http://www.concentric.net/~bereza/checkers/
[3] http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=6366&lngWId=3
[4] http://sourceforge.net/projects/glasscheckers/



Obrázek 4.1: Ukázka programu Glass Checkers
4

Kapitola 5

Návrh implementace vyhovující požadavkům na vyrovnanou hru

Nenašel jsem žádnou literaturu popisující, za jakých podmínek hry je hra dáma vyrovnaná. Proto jsem se rozhodl navrhnout implementaci, kde bude možné hrát podle pravidel hry, podle kterých hrají různé federace dámy. Hra bude mít předvolby, kterými si hráč vybere, podle kterých pravidel chce hrát.

5.1 Návrh tříd aplikace

Jednotlivé třídy jsem navrhnul tak, aby každý objekt odpovídal objektu reálného světa a měl jeho vlastnosti a schopnosti. Diagram tříd můžete vidět na obrázku 5.1. Vytvořil jsem třídu pro hru, která se hraje na hrací ploše. Ta je dalším třídou aplikace. Hrací plocha se skládá z políček. Na hrací ploše jsou umístěny figury. Figura může být kámen nebo dáma. V třídě stav hry ukládá figury a jejich rozmístění na hrací ploše. Kterýkoliv uložený stav hry obsahuje kompletní informaci o rozestavení figur a je možné je kdykoliv zobrazit na hrací ploše. Dále aplikace obsahuje objekt typu tah. Tah se může skládat z více podtahů (pohybů), např. při vícenásobném skoku. Jednotlivé třídy budou v následujících kapitolách. Pro implementaci jsem zvolil jazyk Java, který je vhodný pro implementaci podle objektově-orientovaného návrhu.

5.2 Návrh paralelního algoritmu výpočtu

Hlavní program vytvoří několik vláken, která budou vyhodnocovat jednotlivé větve stromu pro výběr nejvhodnějšího tahu. První volné vlákno bude zpracovávat jednu větev stromu. Další větev bude paralelně zpracovávat další vlákno. Musí být zajištěn výlučný přístup vláken ke stavům zpracovávaných větví stromu, aby se zamezilo případným kolizím vláken. Jakmile hlavní program zjistí, že jsou všechny větve stromu zpracovány, tak vlákna uspí a vrátí nevýhodnější tah.

5.3 Návrh ohodnocovací funkce

Ohodnocovací funkce má za úkol jednoznačně ohodnotit jakýkoliv stav hry podle rozestavění figur na hrací ploše. Ohodnocení (skóre) je pro větší přesnost desetinné číslo.

Základní ohodnocovací funkce spočte počet figur obou hráčů. Počty figur mezi sebou vydělí a výsledek vrátí jako skóre. Jedná se o nejjednodušší ohodnocovací funkci, která se nicméně jeví jako poměrně silná, jelikož se snaží vzít co největší počet figur protihráče a obětovat co nejmenší počet figur hrajícího hráče.

5.4 Podrobnější návrh tříd aplikace

5.4.1 Třída hra (`Game`)

Třída hra je hlavní třídou aplikace. Řídí chod hry a obsahuje veškeré informace o probíhající hře. Tato vlastnost je důležitá. Mám jedno místo, odkud řídím celou hru, a ostatní části aplikace nemohou do hry žádným způsobem zasahovat a narušovat chod hry. Jediným způsobem, jak přerušit hru, je pouze pomocí stisku tlačítka start hry, které ukončí probíhající hru a spustí novou hru, nebo ukončením aplikace. Objekt hra přijímá informace, které nastávají v ostatních modulech, a ty si poté sám zpracovává. Hra se proto vždy nachází v konzistentním stavu.

Třída ovládá grafické uživatelské rozhraní a přijímá od něj informace o akcích, které uživatel provedl na hrací ploše nebo v menu aplikace. Dále třída hra odkazuje na hrací plochu. Hrací plocha je složená z hracích polí a informací týkajících se těchto polí. Každé pole má svou souřadnici na hrací ploše a svou barvu.

Hra se skládá z několika informací o právě probíhající hře, které jsou ve třídě zastoupeny jako atributy. Je to proměnná aktuální stav hry (typ `GameState`), barva aktuálně hrajícího hráče (typ `ColorPlayer`), typ hry (typ `TypeGame`), aktuálně prováděný tah (typ `Turn`), počet řad kamenů každého hráče na začátku hry a nakonec dva seznamy pro variantu hry používající vlákna. Jeden pro uložení vláken a druhý pro uložení úkolů, které budou vlákna zpracovávat.

Aktuální stav hry obsahuje veškeré informace o stavu hry. Ukládá všechny figury rozestavěné na hrací ploše a informace každé z nich. Jde o tyto informace: typ figury (kámen nebo dáma), pozice figury na hrací ploše a barva figury.

Typ hry znamená, jakých typů jsou jednotliví hráči. Hráč může být buď člověk, nebo počítač. Typ hry potom označuje tyto čtyři typy, kdy dva hráči stejných typů hrají proti sobě – počítač-počítač nebo člověk-člověk – nebo dva hráči různých typů hrají proti sobě – člověk-počítač nebo počítač-člověk. Určení všech možných typů hry umožňuje aplikaci jednoznačně zjistit, jaká hra se hraje a aplikace také zná všechny typy hry, které mohou nastat.

Jakmile je známý tah, který se bude hrát, čeká se, až bude proveden a zobrazen na hracím poli pomocí grafického uživatelského rozhraní. Po zobrazení tahu dojde k přepnutí aktuálně hrajícího hráče, který dokončil svůj tah a hraje druhý hráč.

Počet řad kamenů každého hráče na začátku hry označuje, kolik řad kamenů si na začátku hry každý hráč rozestaví. Počet řad kamenů bývá zpravidla určen pravidly, ale je možné, že se hráči dohodnou na jiných pravidlech a rozestaví si větší nebo menší počet řad figur.

Po vytvoření nové hry se inicializují všechny části aplikace, se kterými hra spolupracuje. Jakmile je hra inicializována, spustí se hlavní cyklus aplikace.

5.4.2 Třída hrací plocha (`PlayingBoard`)

Hrací plocha se skládá z hracích polí. Hrací plocha zná svoji velikost. Ukládá všechna hrací pole. Umí vykreslit jednotlivá pole vedle sebe, podle jejich pozice, barvy a velikosti. Hrací plocha umí na svém povrchu vykreslit libovolný stav hry pomocí metody `paint()`. Pokud se jedná o hrací pole, zkontroluje, jestli se na něm nenachází nějaká figura. V případě obsazenosti hracího pole vykreslí odpovídající figuru. Po vytvoření nové hrací plochy se provede její inicializace. Ta spočívá ve vytvoření hracích polí. Hrací pole jsou uložena v dvourozměrném poli objektů typu `Field` pro snadnou indexaci hracího pole. Počet hracích polí je určen velikostí strany hracího pole, kterou je nutno zadat při vytváření hrací plochy. Po vytvoření polí se provede jejich inicializace, tzn., nastavení všech atributů každého pole. Barva pole alternuje u sousedních buněk mezi světlou barvou a tmavou barvou.

5.4.3 Třída hrací pole (`Field`)

Hrací pole je pole zpravidla čtvercového tvaru, které je součásti hrací plochy nebo hrací desky. Hrací pole zná svou pozici na hracím poli a pozici na vykreslovací ploše aplikace. Zná svou barvu a obsahuje příznak, zda je určeno pro hru nebo ne, tzn., jestli se na něm může objevit figura. Obvykle se hra dáma hraje na tmavých polích. Pole se umí vykreslit na příslušné pozici, v dané barvě a velikosti dle atributů pole.

5.4.4 Třída stav hry (`GameState`)

Stav hry je složen z figur rozmístěných na hrací ploše. Stav hry proto ukládá všechny figury a má uloženy všechny jejich atributy. Atributem figury je především pozice, barva a typ figury. Třída stav hry umí rozestavět figury na hrací plochu na začátku hry podle délky strany hrací plochy a podle počtu řad kamenů, které mají být na hrací ploše rozestavěny.

Stav hry můžeme ohodnotit pomocí ohodnocovací metody, která vrátí skóre, které odpovídá aktuálnímu rozestavění figur. Metoda počítá, jak s počtem figur obou hráčů, tak s jejich pozicí na hrací ploše.

5.4.5 Třída figura (`Figure`)

Třída figura je abstraktní třída, která je zobecněním pro třídy kámen a dáma. Figura obsahuje společné vlastnosti pro tyto dvě třídy, které poté přidávají své vlastní implementace vlastností konkrétní figury jim specifické.

Figura má svou barvu a svou pozici na hrací ploše. Figura se umí vykreslit na dané pozici na hrací ploše. Výpočet pozice figury je relativní k pozici hracího pole na vykreslovací ploše aplikace. Ukládá se vždy pozice levého horního rohu objektu (zde pozice hracího pole), od kterého se vykresluje příslušná oblast daných rozměrů. Pro výpočet pozice figury musíme připočíst posun vzhledem k pozici hracího pole, abychom dosáhli umístění figury na střed hracího pole.

Figury se pohybují po hrací ploše vždy diagonálně. Avšak směr pohybu se liší podle typu figury a také podle pravidel hry.

Figura umí zjistit všechny své možné tahy na hrací ploše. Ze stavu hry zná svou pozici a pozice všech okolních figur. Může tedy vyhodnotit a vrátit všechny své možné tahy.

5.4.6 Třída tah (`Turn`)

Tah je elementární třída pro uložení tahu jedné figury. Tah se může skládat z několika podtahů, které jsem nazval pohyby, např. při vícenásobném skoku přes více figur. Tah ukládá seznam těchto pohybů seřazených tak, jak jdou po sobě. Obsahuje také seznam přeskočených figur. To je důležité, pokud hrajeme podle pravidel, kdy se dáma může pohybovat o více polí. Při provádění skoku dámou by potom dáma mohla skočit jednu figuru vícekrát, což nechceme. Proto si ukládáme seznam přeskočených figur, a pokud se nějaká figura nachází v tomto seznamu, znovu ji nemůžeme skočit. Třída tah záměrně neobsahuje odkaz na figuru. Není to nutné, protože figuru, se kterou provádíme tah, můžeme zjistit z počáteční pozice tahu.

5.4.7 Třída pohyb (`Movement`)

Pohyb představuje podtah tahu. Tuto třídu bylo důležité vytvořit zejména kvůli vícenásobným skokům, kdy jeden pohyb představuje přeskočení jedné figury. Hráč provede jeden podtah, ale jeho tah nekončí. Pokračuje v dalších podtazích, dokud vícenásobný skok nedokončí. Až v této chvíli tah končí. Třída pohyb odpovídá reálnému objektu a je v této situaci nezbytná.

Může nastat situace, kdy hráč má možnost provést vícenásobný skok různými způsoby. V průběhu provádění vícenásobného skoku existuje situace, kdy může rozhodnout, kterou figuru skočit. V této situaci je nutné rozlišit obě možnosti a počítat s nimi jako se dvěma různými tahy. Oba tahy obsahují různou sekvenci podtahů, jedná se tedy o různé tahy.

5.4.8 Grafické uživatelské rozhraní (`Gui`)

Grafické uživatelské rozhraní (Graphical User Interface – dále jen GUI) jsem rozdělil do dvou tříd – `Gui` a `GuiMainArea`. Třída `Gui` obsahuje rámec aplikace (frame), menu, lištu menu, stavový řádek a plátno pro vykreslení hrací plochy.

Rozbalovací menu je členěno do pěti částí: *General*, *Game type*, *Rules*, *Search method* a *Parallelism*. Menu *General* obsahuje tlačítko pro start hry a tlačítko pro ukončení aplikace. V menu *Game type* si uživatel zvolí, jaký typ hry bude hrát. Má volbu počítač proti počítači, člověk proti počítači, počítač proti člověku a člověk proti člověku, přičemž vždy první hráč je bílý a druhý černý. V nabídce *Rules* má uživatel možnost zvolit pravidla hry, podle kterých se bude hrát. Vybírá z možností anglická dama, česká dáma nebo si může vytvořit vlastní pravidla dle následujících voleb. Může nastavit velikost hrací plochy 8x8 nebo 10x10, počet řad figur každého hráče na začátku hry nebo může nastavit volby, kdy dáma musí skákat za figurku, nebo dáma se může pohybovat v jednom tahu o více než jedno pole. V dalším menu se nastavuje metoda pro prohledávání stavového prostoru pro případ hry, kdy hraje počítač. Uživatel má na výběr z možností: metoda Alfa-Beta, metoda Minimax využívající vláken, metoda Minimax bez použití vláken. Poslední nabídka umožňuje nastavit počet vláken pro

metodu Minimax, která používá vlákna. Můžeme ponechat automatickou volbu. Potom se vytvoří takový počet vláken, kolik je počet jader procesoru na stroji, na kterém program běží. Nebo můžeme nastavit volbu vlastní a nastavit počet vláken ručně vložením čísla do následujícího pole. Implicitně je v textovém poli vloženo číslo, které odpovídá počtu jader procesoru.

Obrázek 5.1: Diagram tříd

Kapitola 6

Implementace hry

6.1 Proces provádění hry

Grafické uživatelské rozhraní ovládá celý program a zastřešuje interakci s uživatelem. Interakce zahrnuje vizualizaci hry a ovládání hry a aplikace uživatelem. Veškeré informace programu se nastavují pomocí GUI.

Vytvoří se nové objekty pro grafické rozhraní (`Gui`) a hru (`Game`). Následuje jejich inicializace a spuštění. Inicializace hry se skládá z vytvoření hrací plochy, vytvoření proměnné pro uložení aktuálního tahu včetně inicializace seznamu pro podtahy tahu, vytvoření nového stavu hry, vytvoření a rozestavení figur a jejich uložení do stavu hry.

Inicializace GUI zahrnuje vytvoření všech částí GUI jako je rámec aplikace, menu, lišta menu, stavový řádek a plátno pro vykreslení hrací plochy.

V případě, že je aplikace nastavena pro používání dalších vláken, následuje vytvoření a inicializace vláken a proměnných, které vlákna používají. Tato akce bude vysvětlena v kapitole týkající se paralelizace. Nyní jsou všechny části aplikace inicializovány a můžeme spustit samotnou hru. To se zajistí invokací metody `processGame()` ve třídě `Game`.

6.2 Hlavní cyklus hry

Hlavní cyklus hry je cyklus, který provede jeden tah hry a po odehrání tahu zkontroluje stav hry. Odehrání jednoho tahu hry se uskuteční v metodě `play()`. Pokud je stav hry jiný než `OK`, tak se cyklus ukončí, hra skončí a do stavového řádku se vypíše výsledek hry, se kterým hra skončila. Hra může být ukončena výhrou bílého nebo černého hráče, nebo remízou. Existuje ještě jedna situace, kdy může být hra ukončena, a to po stisku tlačítka start hry. Stisk tlačítka je kontrolován po odehrání každého tahu, tedy při každé obrátce cyklu. Po ukončení hry končí i metoda `processGame()`. Z této metody se vracíme do metody `main()`, kde se nachází cyklus, který kontroluje stisk tlačítka start hry. Po stisku tohoto tlačítka se vytvoří a spustí nová hra. Provede se to stejné, co na začátku metody `main()`, akorát některé části, které jsou již inicializovány, není potřeba inicializovat znovu. Proto je jejich inicializace vynechána.

6.2.1 Metoda `play()`

Metoda `play()` vykonává jeden tah hry. Na začátku metody je na stavový řádek napsáno, který hráč je na tahu. Následuje provádění tahu hráčem. Zde se metoda větví na dvě části. V první vykonává tah počítač, ve druhé vykonává tah člověk. Vybere se varianta podle toho, který typ hráče právě hraje. Po odehrání tahu hráče, je nastaven jako aktuálně hrající hráč druhý hráč a hraje protivník. Předtím než protihráč začne hrát, se ale ještě zkontroluje, zda některý hráč nevyhrál nebo jestli nenastala remíza. Pokud žádná z těchto situací nenastala, hra pokračuje.

Výhra některého z hráčů nastane v situaci, kdy jeho protivník nemá na hrací ploše žádné figury. Remíza nastane, pokud některý z hráčů nemůže zahrát žádný tah, ale stále má nějaké figury. To se zjistí tak, že se vygenerují všechny možné tahy hráče, pokud je množina možných tahů prázdná, tak nastane remíza.

Kapitola 7

Podrobný popis důležitých částí aplikace

7.1 Generování tahů figury

Třída figura implementuje metodu, která zjistí všechny možné tahy figury na hrací ploše. Metoda vytvoří prázdný seznam pro uložení možných tahů a zjistí všechny směry, kterými může figura táhnout. Např. bílý kámen se může pohybovat pouze dopředu vlevo a dopředu vpravo. Naopak dáma se může pohybovat všemi směry. Určení směrů pohybu figury umožňuje postupně v cyklu procházet všechny směry a testovat pozice ve směru pohybu na možné tahy figury. Dále se v metodě volá rekurzivní metoda `findTurns()`, která tvoří hlavní část při hledání možných tahů figury.

7.1.1 Metoda `findTurns()`

Na začátku se zjistí počáteční pozice tahu. Ta se vezme buď z pozice figury, pokud provádím první pohyb figurou, nebo z posledního podtahu tahu uloženého v proměnné `aTurn`. Následuje hlavní cyklus `for`, který všemi směry, kterými se může figura pohybovat, prochází hrací plochu a hledá její možné tahy. Uvnitř cyklu `for` se nachází další cyklus `while`, který pro posun figury o daný počet polí ve směru pohybu, který je nastaven v cyklu `for` (např. doleva dopředu o dvě pole). Kámen se zpravidla pohybuje o jedno pole, proto po otestování pohybu o jedno pole cyklus `while` ukončím a pokračuji v procházení dalšího směru. Stejně tak při pohybu dámou v anglické dámě, kde se může dáma pohybovat pouze o jedno pole.

Pokud chci provádět nový tah, vytvořím novou prázdnou proměnnou, do které tah uložím. V případě, že jsem se rekurzivně zanořil pro hledání vícenásobného skoku, vytvářím kopii tahu a všech jeho podtahů, které jsem předtím provedl, protože může nastat více možností vícenásobného skoku a já chci každou možnost uložit jako jiný tah.

Nyní zpět k pohybu ve směru. Metoda `goInDirection()` mi vrátí pohyb (`Movement`), pokud jí řeknu počáteční pozici, směr a o kolik polí. Poté pracuji pouze s proměnnou tohoto pohybu a v případě, že je pohyb platný, uložím jej do seznamu možných tahů. Ale k tomu až později. Zkontroluji, jestli se po provedení tohoto podtahu nebudu nacházet mimo hrací pole. Takový tah by byl neplatný. Poté zkontroluji, jestli pole, na které se chci přesunout, není obsazené. V případě, že ne, je tah platný a přidám jej do množiny možných tahů. Pokud je pole obsazené, zjistím, jestli není obsazené figurou protihráče. V takovém případě

by mohla nastat možnost figuru skočit. Provedení skoku by však bylo možné jen v situaci, kdy by za skákanou figurou bylo volné místo. To je potřeba otestovat. Pozice se otestuje, zda leží uvnitř hrací plochy a jestli není obsazena. Pokud je pozice volná, jedná se o platný tah. Podtah je přidán do seznamu podtahů v tahu, ale tah ještě není do množiny možných tahů. Nejdříve je rekurzivně volána metoda `findTurns()`, uvnitř které se zjistí, zda není možné provést další podtah formou vícenásobného skoku. Jakmile žádný další podtah není možné přidat, je teprve potom celý tah přidán do množiny možných tahů.

Zajímavou částí je přidávání tahu do seznamu tahů na základě proměnné `aWasAdded`. Tato proměnná byla vytvořena z důvodu přidávání tahu do seznamu možných tahů v situaci, kdy jsme provedli skok a hledáme další skok, který na něj navazuje. Žádný další skok nenalezneme a tah je ukončen. Jenže tah je potřeba vložit do seznamu tahů. Tah vložíme do seznamu možných tahů, jakmile nemůže být k tahu přidán žádný další podtah. To je v okamžiku, kdy jsme prošli všechny směry, kterými se může figura pohybovat a ani v posledním směru nebyl žádný tah přidán. Zároveň musí být proměnná nastavena na hodnotu `false`. To znamená, že tah předtím nebyl přidán do seznamu tahů, a proto jej tam přidáme nyní. Proměnnou `aWasAdded` nastavuji vždy na hodnotu `false` před voláním rekurze, tj. po provedení jakéhokoliv skoku. Poté je předána při rekurzivním volání funkce jako parametr.

Po vynoření z rekurze následuje další cyklus. Ten je však programem proveden pouze v případě, že je v nastavení hry povoleno, aby hráč nemusel dámou skákat přímo za figuru, ale aby mohl zaujmout libovolné místo za skočenou figurou. Nacházíme se v situaci, kdy jsme skočili figuru. Prohledáváme postupně místa za přeskočenou figurou. Pokud najdeme volné místo, jedná se nový možný tah, který musíme korektně zařadit do seznamu možných tahů. Zároveň opět rekurzivně otestujeme, jestli z cílové pozice tohoto skoku nemůže nastat další skok a nejedná se o vícenásobný skok.

7.2 Procházení stavového prostoru

Procházení stavového prostoru probíhá pomocí algoritmu Minimax a algoritmu Alfa-Beta. Uživatel aplikace má možnost si vybrat, kterou metodu bude aplikace používat.

7.2.1 Metoda Minimax

Metoda Minimax se v programu nazývá `generateTurnMinimax()`, protože generuje nejlepší tah za pomoci metody Minimax. Je to rekurzivní metoda. Vrací objekt třídy `TurnScore`. Tato třída slouží pro uložení nejlepšího nalezeného tahu a jeho ohodnocení označeného jako skóre. Skóre je pro větší přesnost desetinné číslo datového typu `double`. K ničemu jinému se tato třída nepoužívá.

Pokračuji v popisu metody `generateTurnMinimax()`. Pokud se nacházíme v listových uzlech (hodnota hloubky `aDepth` je rovna nule), tak ohodnotíme stav hry v lisovém uzlu a vrátíme skóre pomocí proměnné `lTurnScore`. V této situaci v proměnné nevracíme žádný tah. V ostatních případech, kdy jsme ještě nedorazili k listovým uzlům, generujeme možné tahy pro všechny figury hráče hrajícího v dané úrovni stromu. Jakmile máme všechny možné tahy hráče vygenerované z daného stavu hry. Odehrajeme postupně každý z nich nad stavem hry a vytvoříme nový stav hry, který obsahuje figury rozestavěné tak, jak by byly umístěné na hrací ploše po odehrání tahu. Tzn., figura hrající tah bude na cílové pozici

tahu a figury, které byly skočeny, jsou odstraněny. Nyní už pracujeme pouze s tímto stavem hry. Rekurzivně voláme metodu `generateTurnMinimax()`, kdy parametrem je nový stav hry a hloubka snížená o hodnotu jedna. Dalším parametrem je hráč, který aktuálně v dané větvi stromu hraje. Po zanoření ve stromu hraje druhý hráč, který je nastaven jako parametr. Ostatní parametry zůstávají stejné jako při prvním volání metody. Rekurzivně volaná metoda vrátí objekt typu `TurnScore`, odkud vezmeme nejlepší nalezené skóre. Tah, pro který je skóre určeno, známe. Vytvářeli jsme s ním nový stav hry, který jsme poté předávali do metody. Nyní porovnáváme skóre s dosud nejlepším nalezeným skóre. Pokud jsme ještě žádné skóre nezaznamenali, uložíme si tah a skóre. V případě, že už nějaké skóre máme, porovnáváme s ním nově získané skóre. Pokud je nové skóre lepší, uložíme jej jako dosud nejlepší nalezené skóre a přidáme k němu tah. Při porovnávání skóre hledáme pro právě hrajícího hráče tahy s nejvyšším skóre a pro jeho protihráče hledáme tahy s co nejmenším skóre. Odtud název metody Minimax.

Rozšířením metody Minimax je doplněním náhodného výběru tahů, pokud máme více tahů se stejným skóre.

7.2.2 Metoda Alfa-Beta

Metoda AlfaBeta je podobná metodě Minimax. Obsahuje však rozšíření, které dramaticky ovlivní rychlost vyhledání nejlepšího tahu. Metodu implementuje metoda `generateTurnAlphaBeta()`. Následující části jsou stejné jako u Minimaxu. V listových uzlech se vrací ohodnocení stavu hry. Pokud nejsme v listech, generujeme tahy jednotlivých figur hráče a nad nimi rekurzivně voláme metodu `generateTurnAlphaBeta()` s hloubkou sníženou o jedna, nově vzniklým stavem hry a jako hrajícího hráče nastavíme protihráče. Dosud bylo vše stejné, ale teď přichází změna. Navíc metodě předáme parametry Alfa a Beta, což jsou desetinná čísla. Slouží k detekci Alfa nebo Beta řezů a zároveň pro uchování nejlepšího skóre hráče A, resp. hráče B. Metoda nám vrátí nejlepší nalezené skóre, které vyhodnotíme. Hraje-li hráč A (hrající hráč), je nově nalezené skóre porovnáváno s hodnotou Alfa. Pokud je skóre větší, jedná se o nově nalezené nejlepší skóre a je přidáno do objektu `lTurnScore` a hodnota Alfa je nastavena na hodnotu nově nalezeného nejlepšího skóre. Hledáme větší hodnotu než Alfa, protože pro hráče A hledáme jeho nejlépe ohodnocené tahy. Naopak hraje-li hráč B (protihráč hrajícího hráče), je hodnota nově nalezeného skóre porovnávána s hodnotou proměnné Beta. Pokud je nově nalezené skóre menší než hodnota Beta, přidáme jej do objektu `lTurnScore` a hodnotu Beta nastavíme na hodnotu nového skóre. Analyticky hledáme hodnotu menší než Beta, protože pro hráče B hledáme co nejhorší tah. Navíc je pro hráče A i pro hráče B přidána podmínka, která pokud je splněna, nastane Alfa nebo Beta řez. Strom se od této části dál nevyhodnocuje, čímž se ušetří čas prohledávání této části stavového prostoru. Podmínka Beta řezu je splněna, hraje-li hráč A a pokud je hodnota Alfa menší nebo rovna hodnotě Beta. V této situaci je vrácena hodnota Alfa. Podmínka Beta řezu je splněna, hraje-li hráč B a pokud je hodnota Alfa menší nebo rovna hodnotě Beta. Podmínka je stejná jako u hráče A. Rozdíl je v tom, že se vrací hodnota proměnné Beta. Pokud nenastal ani Alfa, ani Beta řez, vrací se hodnota Alfa, když hraje hráč A. Hraje-li hráč B, vrací se hodnota Beta.

Do metody `generateTurnAlphaBeta()` byla přidána náhodnost stejným způsobem jako u metody `generateTurnMinimax()`.

7.3 Ohodnocování stavu hry

Ohodnocení stavu hry se dělí na dvě části. První je tzv. **materiální ohodnocení**, které se dívá pouze na počet figur jednotlivých figur obou hráčů. Druhé ohodnocení je **poziční ohodnocení**, které zohledňuje rozestavění a pozice figur na hrací ploše.

Materiální ohodnocení spočítá počet figur hrajícího hráče a počet figur jeho protihráče. Přičemž dáma má dvojnásobnou hodnotu než kámen. Jakmile má figury obou hráčů sečteny, vydělí oba výsledky mezi sebou a získá materiální ohodnocení.

Poziční ohodnocení se vypočítá u kamene jako číslo řady na šachovnici plus 5, jako ohodnocení kamene. Číslo řady, protože chceme zvýhodnit pozice, které jsou blíže opačnému konci hrací plochy, a tzn., nasazení dámy. Dáma má ohodnocení 7 plus maximální ohodnocení pozice, tzn. délka strany hrací plochy, protože u dámy nepotřebujeme zvýhodňovat pozici. Dámou se nepotřebujeme dostat na žádný konec hrací plochy. [2]

Kapitola 8

Paralelní algoritmus výpočtu

Paralelní algoritmus jsem navrhl pro algoritmus Minimax pro zvýšení rychlosti výpočtu vyhodnocování stromu. Strom nemusí být pomocí algoritmu Minimax vyhodnocován sekvenčně, proto lze paralelizaci dobře využít.

Paralelní algoritmus jsem nenavrhoval pro algoritmus Alfa-Beta, protože navrhnout takový algoritmus by bylo velmi komplikované. Algoritmus Alfa-Beta je sekvenční algoritmus. Tzn., že strom musí být vyhodnocován postupně (sekvenčně) a začlenit paralelizaci do výpočtu by bylo složité. Navíc by bylo pravděpodobné, že některé větve stromu by paralelní algoritmus vyhodnocoval zbytečně. Metoda by mohla vyhodnocovat paralelně nějakou větev stromu, nad kterou by nastal Alfa nebo Beta řez a větev by byla vyhodnocena zbytečně.

O návrh paralelního algoritmu pro metodu Alfa-Beta se pokusili T. A. Marsland a M. Campbell z University of Alberta v Edmontonu v Kanadě ve své práci *Parallel Search of Strongly Ordered Game Trees* [6]. Další, kdo vytvářel paralelní algoritmus pro tuto metodu, byli Geoffrey C. Fox, Roy D. Williams a Paul C. Messina ve své práci *Parallel computing works!* [3] Oba způsoby paralelizace byly však natolik složité, že jsem od paralelizace pro metodu Alfa-Beta ustoupil a navrhl jsem pouze paralelizaci pro metodu Minimax.

Už jen pro zajímavost uvádím návrhy paralelní metody Alfa-Beta pro hru šachy. První varianta používá paralelně-sekvenční výpočet pomocí metody Principal Variation Splitting (PVS). Publikace je od Briana Greskampa z univerzity v Illinois a nazývá se *Parallelizing a Simple Chess Program* [4]. Druhá publikace popisuje řešení pomocí metod Event Split a Master/Slave. Výsledky obou metod jsou přehledně zobrazeny pomocí grafů. Jedná se o publikaci *Parallel Alpha-Beta Pruning of Game Decision Trees: A Chess Implementation* [7] od Kevina Steeleho z Brigham Young University.

8.1 Implementace paralelního algoritmu

Hlavní vlákno programu nejdříve vytvoří seznam pro úkoly, které budou vlákna zpracovávat. Následně jsou vytvořena a spuštěna vlákna, která v pravidelných intervalech kontrolují seznam úkolů. Implicitně je vytvořeno tolik vláken, kolik je procesorů v systému. Ale uživatel má možnost si nastavit počet vláken podle sebe. Jakmile se v seznamu objeví nějaký nezpracovaný úkol, vlákno si jej odebere a zpracuje jej. Nyní je seznam úkolů prázdný, takže vlákna nemají co zpracovávat. Jakmile začne hra počítače a počítač má vygenerovat nejle-

pší možný tah, přijdou na řadu vlákna. Hlavní program vytvoří úkoly, které budou vlákna zpracovávat. Vytvoření úkolů má na starosti metoda `generateTurnMinimaxMainThread()`. Metoda nejdříve vygeneruje tahy všech figur hrajícího hráče. Prohledá vygenerované tahy, a pokud nalezne nějaké tahy, které provádějí skok protihráčovy figury, tak odstraní všechny neskákací tahy z důvodu povinnosti skákání. Každý z nalezených tahů bude jedním úkolem pro vlákna a nakonec se z těchto tahů vybere jeden tah, který je nejlepší.

Nyní se metoda snaží získat výlučný přístup k seznamu úkolů, které budou provádět vlákna, aby přidala jejich nové úkoly. To zajistíme pomocí volání metody `synchronized()` nad seznamem úkolů, který nastavíme jako parametr metody. Jakmile získáme přístup k seznamu, přidáváme nové úkoly do seznamu. Úkol vlákna musí obsahovat všechny informace, které vlákno potřebuje, aby mohlo začít úkol zpracovávat, a musí zde mít také proměnnou, kam uloží výsledek výpočtu, který provádí. Úkoly jsou objekty třídy `ThreadTask`. Třída obsahuje stav úkolu, který může nabývat hodnot nezpracovaný, zpracovávaný a zpracovaný. Dalším atributem třídy je proměnná třídy `TurnScore` pro uložení výsledku výpočtu, který provedlo vlákno. Následují parametry pro metodu `generateTurnMinimax()`, kterou vlákno `MinimaxThread` používá pro vyhledávání nejlepšího možného tahu.

Jakmile jsou hlavním vláknem vytvořeny všechny úkoly. Hlavní vlákno (proces) opustí synchronizovanou část programu a nechá běžet vlákna. První vlákno dostane výlučný přístup k seznamu úkolů a začne jej prohledávat. Hned na začátku seznamu úkolů najde nezpracovaný úkol. Ten si odebere a označí jej jako zpracovávaný. Poté opustí kritickou sekci, aby další vlákna mohla zpracovávat další úkoly. Vlákno začne zpracovávat úkol, a jakmile jej dokončí, uloží výsledek do objektu úkolu `ThreadTask`. Poté vlákno požádá o přístup k seznamu úkolů. V okamžiku, kdy přístup dostane, označí úkol jako zpracovaný a opustí kritickou sekci. Poté vlákno v cyklu pokračuje stejným způsobem jako předtím, dokud nejsou všechny úkoly zpracované.

Hlavní vlákno v pravidelných intervalech kontroluje seznam úkolů, zda již nejsou všechny úkoly zpracovány. Přičemž dodržuje výlučný přístup k seznamu mezi všemi vlákny. Jakmile zjistí, že jsou všechny úkoly hotovy, začne zpracovávat výsledky, které vypočítala vlákna. Pomocí `sychronized()` zablokuje seznam výsledků, aby jej vlákna zbytečně nekontrolovala na nové úkoly, a prochází zpracované výsledky. Z vypočtených výsledků hledá výsledek, který má nejvyšší skóre. Tento výsledek je spolu s tahem vrácen jako nejlepší nalezený tah pro hráče. Před ukončením metody je vyprázdněn seznam úkolů pro vložení dalších nových úkolů.

Následuje popis třídy `MinimaxThread`, která obsahuje implementace metod vláken pro výpočet jednotlivých větví rozhodovacího stromu. Třída `MinimaxThread` dědí třídu `Thread` a zachovává většinu vlastností této třídy. Některé metody má však přepsané a některé doplněné. Přepsaná je metoda `run()`, což je metoda, která se provede po spuštění vlákna. Nastavení vlákna do stavu běžící a jeho spuštění se provede pomocí metody `start()`. Poté se zavolá metoda `run()`, která obsahuje hlavní cyklus vlákna. Jedná se o nekonečný cyklus, ve kterém vlákno zkontroluje, jestli se v seznamu úkolů pro vlákna neobjevil nějaký nezpracovaný úkol. Pokud nějaký úkol vlákno najde, tak jej zpracuje. Poté, co jej vlákno zpracovalo, pokračuje tím, že se na nějakou dobu uspí a následuje další obrátka nekonečného cyklu.

Vyhledávání nezpracovaného úkolu provádí metoda `appearsUnprocessed()`, která vrací hodnotu `true`, jakmile najde nějaký nezpracovaný úkol. Metoda musí získat výlučný přístup k seznamu úkolů. To zajistí metoda `synchronized()`. V synchronizovaném bloku kódu prohledá celý seznam. V případě, že nalezne nějaký úkol ve stavu nezpracovaný, vrátí `true`. Pokud projde celý seznam a žádný takový úkol nenajde, vrátí `false`. Metoda `appearsUnprocessed()` je navíc označena jako `synchronized`, aby se zbytečně nepokoušelo více vláken vyhledávat v seznamu nezpracovaný úkol.

Poslední metodou třídy `MinimaxThread` je metoda `process()`, která zpracovává nezpracovaný úkol. Metoda požádá o výlučný přístup k seznamu úkolů a vyhledá první nezpracovaný úkol. Lokálně si uloží referenci na tento úkol a úkol nastaví do stavu zpracovávaný. Tím zajistí, že žádné jiné vlákno nebude úkol zpracovávat. Poté vlákno opustí kritickou sekci a umožní přistupovat k seznamu úkolů ostatním vláknům. Nyní vlákno začne zpracovávat samotný úkol. Vlákno zavolá metodu `generateTurnMinimax()`, která je statická, aby ji bylo možné volat nezávisle na jakékoliv instanci. Parametry metody jsou uloženy v úkolu, tedy v objektu třídy `ThreadTask`. Metoda `process()` zavolá metodu `generateTurnMinimax()` s těmito parametry a vrácený výsledek uloží opět do zpracovávaného úkolu. Nakonec požádá o výlučný přístup k seznamu úkolů, a jakmile jej získá, nastaví úkol do stavu zpracovaný. V tuto chvíli metoda `process()` končí a vlákno pokračuje v hlavním cyklu.

8.2 Vyhodnocení výsledků obou metod

Následující testování jsem provedl na této konfiguraci systému: Intel Pentium D, CPU 3,20 GHz, jádra: 2, 3 GB RAM, základní deska: MSI 945P, GPU: ATI Radeon HD 5670.

Doby výpočtu nejlepšího tahu jednotlivými metodami:

Metoda Alfa-Beta průměrně vypočte jeden tah za 156,6 ms. Metodě Minimax bez použití vláken zabere výpočet jednoho tahu 1207,5 ms a paralelní metodě Minimax, která používá vlákna, zabere výpočet průměrně 971,8 ms. Časy jsou měřeny pro podobné situace hry, kdy mají oba hráči přibližně stejný počet tahů, aby metody procházely podobně velké stavové prostory. Metody výpočtu hrály podle pravidel české dámy se dvěma řadami kamenů. Metoda paralelního Minimaxu používala pro výpočet dvě vlákna a hloubku prohledávání stromu 6. Metodu AlfaBeta bylo možné nastavit pro testování na hloubku prohledávání 8 a její rychlost byla pořád dostačující.

Z výsledků můžeme zjistit, že metoda Alfa-Beta bude vždy rychlejší než metoda Minimax. Použití vláken sice výpočet mírně zrychlilo (přibližně o 250 ms), ale metoda Alfa-Beta je stále několikanásobně rychlejší. Metoda Alfa-Beta je přibližně 7krát rychlejší než Minimax. Při použití vláken je Alfa-Beta 6krát rychlejší než paralelní Minimax.

Při zvětšení hloubky prohledávání stromu u metody AlfaBeta z 8 na 10 a u obou Minimaxů z 6 na 8 se rychlost výpočtu jednoho tahu velmi zvýšila. AlfaBeta vypočetla průměrně tah za 2,7 s, ale u Minimaxu to bylo ještě více. Obyčejná metoda Minimax vypočetla tah za 134 s, paralelní Minimax byl o něco lepší – 102 s.

Metoda Alfa-Beta byla úspěšnější než metoda Minimax. Alfa-Beta vždy skončila výhrou bílého hráče. Metoda Minimax skončila výhrou bílého hráče pouze přibližně ve třetině

případů. Avšak na výhru měla také vliv ohodnocovací funkce. Její vylepšení by dle mého názoru vedlo k zlepšení hry metody Minimax. Testování je detailně popsáno v následující kapitole.

V poslední části vyhodnocení jsem testoval, zda je vždy Alfa-Beta řez proveden správně. Nenalezl jsem žádnou chybu v metodě Alfa-Beta a dle mého názoru provádí Alfa a Beta řezy a prochází stavový prostor správně. Podle výsledků testování této metody, které jsou velmi dobré, můžeme usoudit, že metoda funguje správně.

Kapitola 9

Ověření implementace na sadě testů

Byly otestovány všechny části grafického uživatelského rozhraní. Všechny položky menu jsou funkční. Bylo prokázáno, že nastavení provedená v menu se projeví v aplikaci. Dále byla otestována detekce pokynů uživatele k pohybu figur na hrací ploše a korektní pohyb figur. Bylo ověřeno, že hráč může provádět pouze tahy, které odpovídají pravidlům. Jiné tahy byly zamítnuty a neplatný tah nebyl hráči umožněn.

Bylo otestováno, že hra se vždy dohraje do konce. Hra byla spuštěna pro každou vyhodnocovací strategii v nekonečné smyčce, kdy po skončení jedné hry se ihned spustila další hra se stejným nastavením. Aplikace bez problému odehrála s každou herní strategií 30 minut.

Aplikace byla spuštěna opět v nekonečném cyklu a měřil jsem dobu jednotlivých partií při vypnutí všech zpomalení hry a při nastavení typu hráčů počítač-počítač. Doby hraní jednotlivých hracích strategií jsou následující. Průměrná doba hry při použití metody Alfa-Beta byla 3,7 s. Při použití algoritmu Minimax byla doba jedné partie více než čtyřnásobná, 15,9 s. S využitím paralelizace vláken aplikace hru odehrála za 14,3 s. Toto zrychlení není příliš velké, nicméně zanedbatelné není. Prokazuje, že paralelizace vylepšila rychlost výpočtu. Předpokládám, že na více než dvoujádrovém stroji by byl rozdíl v rychlosti větší.

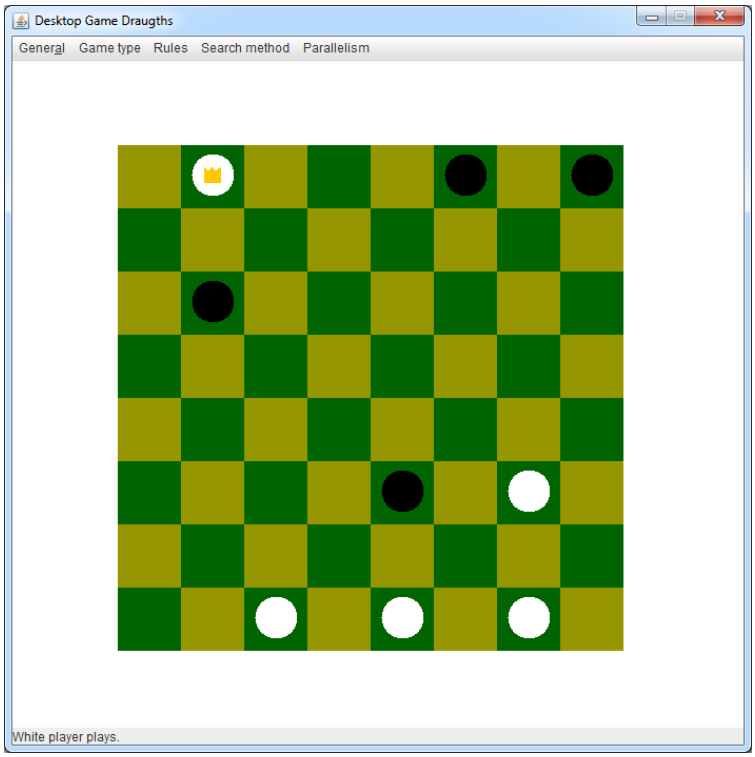
Při použití algoritmu Alfa-Beta hra vždy skončila výhrou bílého hráče. Při použití Minimaxu většinou vyhrál černý hráč (11krát), ale někdy také bílý (5krát) a dvakrát nastala remíza. Počet výher bílého a černého hráče byl shodný při použití paralelního Minimaxu.

Poslední částí testování bylo nastavení hry, kdy více strategií evaluace hraje proti sobě. Vyhodnocovací strategie, která používá poziční ohodnocení, v testování nedopadla příliš dobře. Proti jednoduchému materiálnímu ohodnocování vyhrála pouze v 30 % partií, což je velmi málo. Budoucí vylepšení poziční ohodnocovací metody by zcela jistě vedlo k zlepšení hry a poziční ohodnocování by mělo porazit materiální ohodnocovací metodu.

Kapitola 10

Demonstrace aplikace

Aplikace umožňuje hru proti hráče hráči, proti počítači nebo dvou počítačů proti sobě. Je možné nastavit hru podle různých pravidel (česká dáma, anglická dáma nebo vlastní pravidla). Pro hráče typu počítač umožňuje vybrat metodu pro vyhledání nejlepšího možného tahu. Uživatel má na výběr z možností metod Alfa-Beta, Minimax nebo paralelní Minimax, u kterého je možné nastavit počet vláken, které bude algoritmus pro výpočet používat. Aplikace je open-source a je dostupná na internetu[1]. Je vhodná pro budoucí rozšíření.



Obrázek 10.1: Ukázka aplikace

[1] https://sourceforge.net/p/paraldraughts

Kapitola 11

Zhodnocení práce a návrhy jejího rozšíření

Vytvořil jsem funkční a přehlednou aplikaci, která se bude jako open-source dobře rozšiřovat. Je přehledně napsaná a dobře objektově navržená. Aplikace nedosahuje výsledků profesionálních aplikací, nicméně po rozšíření by se jim mohla přiblížit. Implementace paralelního algoritmu je podle mého názoru optimální a dobře navržená. Na vícejádrovém stroji by mohla dosáhnout dobrých výsledků. Program podporuje hru podle pravidel české a anglické dámy, které vyhovují požadavkům na vyrovnanou hru. Navíc uživateli umožňuje vytvořit si svá vlastní pravidla pomocí parametrů hry.

Hra obsahuje implementace algoritmů Alfa-Beta a Minimax, které slouží pro procházení stromů možných tahů a hledání nejlepšího možného tahu pro hrajícího hráče. Program také podporuje hru dvou hráčů-lidí proti sobě nebo člověka proti počítači. Aplikace hlídá, zda jsou jejich tahy platné a v případě pokusu o zahrání neplatného tahu, tento tah nepovolí.

Umělá inteligence počítače je velmi jednoduchá, nicméně pro málo pokročilou hru postačuje. Pomohlo by vylepšení ohodnocovací metody. Metoda by se v různých situacích hry měla chovat jinak. V situaci, kdy má hráč samé dámy by aplikace měla hrát jinak. Měla by také hrát jinak v různých částech hrací plochy. Např. kameny v řadách na začátku, uprostřed a na konci hrací plochy by měly mít jiné ohodnocení. Dalším vylepšením by mohlo být použití jiné ohodnocovací metody v situaci, kdy hrajeme s více dámami. Při hraní dámou zpravidla uplatňujeme jiné strategie než při hraní kameny.

Po implementaci několika ohodnocovacích metod by bylo možné prokázat, že některé metody hrají viditelně lépe. Různé metody by byly přiděleny jednotlivým stupňům obtížnosti hry a inteligence počítače (např. začátečník, středně pokročilý a pokročilý).

Aplikace by také bylo možné rozšířit o síťovou verzi. Uživatelé by spolu hráli dámu po síti. Jeden z hráčů by musel být klient a druhý server, nebo by se oba hráči připojili na server, který by hru zprostředkovával. Komunikace by probíhala pomocí jednoduchého protokolu obsahujícího jednotlivé příkazy, pokyny a informace týkající se hry. Např. informace o odehraném tahu hráče.

Kapitola 12

Závěr

V této práci byla implementována stolní hra dáma, která umožňuje hru podle více pravidel. Tato pravidla používají různé federace dámy a vyhovují požadavkům na vyrovnanou hru. Pomocí nastavení aplikace lze používat různé vyhodnocovací strategie a tím ovlivňovat aspekty hry k lepšímu či horšímu. Hra podporuje paralelní zpracovaní stavového prostoru pomocí metody Minimax.

Hru jsem implementoval podle objektově-orientovaného návrhu v jazyku Java, který vhodný pro multiplatformní aplikace. Aplikace je open-source a je vhodná pro budoucí rozšíření.

Literatura

[1] Oficiální pravidla české a mezinárodní dámy dle ČFD. 2000.
URL http://www.damweb.cz/pravidla/pravidla.html

[2] Cohen, L.: AI Course Final Project – Checkers. 2010.
URL http://www.cs.huji.ac.il/~lirchi/AIP/English-Draughts.pdf

[3] Fox, G. C.; Williams, R. D.; Messina, P. C.: *Parallel computing works!* Morgan Kaufmann Publishers, 1994.
URL http://www.netlib.org/utk/lsi/pcwLSI/text/BOOK.html

[4] Greskamp, B.: Parallelizing a Simple Chess Program, 2003.
URL http://iacoma.cs.uiuc.edu/~greskamp/pdfs/412.pdf

[5] Marek, V.; Kalendovský, J.: *Dáma a šach jako zábava, trénink ducha a sport.* Portál, vyd. 1. vydání, 2001.

[6] Marsland, T. A.; Campbell, M.: Parallel Search of Strongly Ordered Game Trees. *ACM Computing Surveys*, ročník 14, č. 4, 1982: s. 533–551.
URL http://portal.acm.org/citation.cfm?doid=356893.356895

[7] Steele, K.: Parallel Alpha-Beta Pruning of Game Decision Trees: A Chess Implementation. 1999.
URL http://students.cs.byu.edu/~snell/Classes/CS584/projectsF99/steele/report.html

[8] Zbořil, F.; Zbořil, F.: Základy umělé inteligence. 2013.
URL
https://www.fit.vutbr.cz/study/courses/IZU/private/oporaizu-esf-5a.pdf